CHAIR OF DECENTRALIZED INFORMATION SYSTEMS & DATA MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Thesis proposal

Concurrent Range-Locking

Author: **Thua-Duc Nguyen**

Supervisor: **Prof. Dr. Viktor Leis**

Advisor: **Lam-Duy Nguyen**

1 Introduction

In many popular systems, including database, file, and operating systems, acquiring exclusive locks on continuous values is crucial, as highlighted in several studies [1, 2, 3, 4, 5, 6]. One notable implementation of this mechanism is the range lock [5, 7]. Range locks offer a more refined approach than the traditional single-lock technique – by partitioning a shared resource into multiple arbitrary-sized segments, which can be acquired exclusively by different processes. This strategy effectively addresses the drawbacks and bottlenecks associated with single-lock methods, significantly improving the performance.

2 Motivation

Virtual-memory needs range locking. The Linux kernel community recently tried to use range-locking to improve the scalability issue of `mmap_lock` [2, 8, 9]. The `mmap_lock` uses a per-process semaphore to control access to the whole `mm_struct` [10] and serialize changes to address spaces. Despite previous efforts to overcome the scalability issues of `mmap_lock`, a resolution is yet to be found [9].

Range locking in DBMS. As database sizes increase exponentially, locking the entire database becomes impractical. Such a coarse-grained locking mechanism prevents concurrent transactions from progressing, resulting in poor throughput and high latency. The previous key-range locking in DBMS [3, 11] is complex and tightly coupled with lock-based concurrency control protocols [11]. As a result, this technique is not applicable for general DBMS operations – such as variable-sized page allocation – hence, a new technique is desirable.

3 Related Work

Previous research has explored various approaches to range lock [12, 13, 7]. The current implementation of range lock in the Linux kernel uses a range tree that keeps track of acquired ranges and an internal spin lock to protect it [12]. Since every range request relies on this single spinlock, it becomes a point of contention.

Song et al. [13] try to improve the Linux kernel's implementation by combining a skip list with a spinlock to manage locked ranges. This technique leverages the skip list, which is more lightweight and efficient than the interval tree and can still conduct intensive searches for overlapping ranges. Despite these advancements, the problem of contention points still requires resolution.

In another research, Kogan et al. [7] designed a range lock based on a concurrent linked list, where each node represents an acquired range. This design achieves a lock-free mechanism that effectively addresses existing range locks' shortcomings. However, the linked list's insertion and lookup operations are less efficient than tree-like structures.

4 Approach

In this research's scope, we propose a new concurrent range-locking design that leverages a probabilistic concurrent skip list [14, 15]. It consists of two main functions:

- **try_lock**: The `try_lock` function searches for the required range `[start, start+len)` in the skip list. If an overlapping range exists, indicating another thread is modifying that range, the requesting thread must wait and retry. If not, the range is added to the list, signaling that the range is reserved.
- **release_lock**: The `release_lock` function releases the lock by finding the address range in the skip list and removing it accordingly.

Our range lock design also utilizes the per-node lock instead of an interval lock, thus addressing the bottleneck problem of the spinlock-based range lock and maintaining the lock's high level of performance.

5 Evaluation

We will evaluate the proposed approach under these evaluation criteria:

- **Performance:** We will test the range lock mechanism under increasing load and concurrent accesses to measure its performance.
- **Correctness:** We will ensure the consistency and correctness of data accesses, especially when there are overlapping data ranges and concurrent operations.
- **Comparison:** We will compare the performance of the proposed solution with existing state-of-the-art approaches.

6 Expected Outcome

We aim to develop a concurrent range-locking mechanism that performs better than the existing range locks. The evaluation results will provide insights into the proposed mechanism's performance characteristics and potential trade-offs.

7 Resources

We will need 32 cores and 32 GB of RAM for one month. These resources allow us to perform thorough tests under heavy contention and multithreaded scenarios.

Bibliography

[1] D. B. Lomet. *Key range locking strategies for improved concurrency*. Digital Equipment Corporation, Cambridge Research Laboratory UK, 1993.

[2] J. Corbet. "Range reader/writer locks for the kernel". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/724502/.

[3] G. Graefe. "Hierarchical locking in B-tree indexes". In: *On Transactional Concurrency Control*. Springer, 2007, pp. 45–73.

[4] C.-G. Lee, S. Noh, H. Kang, S. Hwang, and Y. Kim. "Concurrent file metadata structure using readers-writer lock". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021, pp. 1172–1181.

[5] J. Gao, Y. Lu, M. Xie, Q. Wang, and J. Shu. "Citron: Distributed Range Lock Management with One-sided {RDMA}". In: *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 2023, pp. 297–314.

[6] L. Chang-Gyu, B. Hyunki, N. Sunghyun, K. Hyeongu, and Y. Kim. "Write optimization of log-structured flash file system for parallel I/O on manycore servers". In: *Proceedings of the 12th ACM International Conference on Systems and Storage*. 2019, pp. 21–32.

[7] A. Kogan, D. Dice, and S. Issa. "Scalable range locks for scalable address spaces and beyond". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–15.

[8] M. Rybczynska. "Introducing maple trees". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/845507/.

[9] J. Corbet. "The ongoing search for mmap_lock scalability". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/893906/.

[10] S. Boutnaru. "Linux Kernel — mm_struct". In: *medium.com* (2023). Accessed: 2024-04-21. URL: https://medium.com/@boutnaru/linux-kernel-mm-struct-fafe50b57837.

[11] A. Pavlo. "Two-Phase Locking". In: *15445.courses.cs.cmu.edu* (2022). Accessed: 2024-04-21. URL: https://15445.courses.cs.cmu.edu/fall2022/slides/16-twophaselocking.pdf.

[12] J. Kara. "Implement range locks". In: *lkml.org* (2013). Accessed: 2024-04-21. URL: https://lkml.org/lkml/2013.

[13] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen. "Parallelizing live migration of virtual machines". In: *Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 2013, pp. 85–96.

[14] H. Maurice, L. Yossi, L. Victor, and S. Nir. "A provably correct scalable concurrent skip list". In: *Conference On Principles of Distributed Systems (OPODIS). Citeseer*. Vol. 103. 2006.

[15] H. Maurice, S. Nir, L. Victor, and S. Michael. *The art of multiprocessor programming*. Newnes, 2020.